```
import pandas as pd
import numpy as np
import sklearn
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.metrics import *
from sklearn.model_selection import *
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files Iris_data.csv

- **Iris_data.csv**(application/vnd.ms-excel) - 4966 bytes, last modified: 6/7/2021 - 100% done

```
Saving Iris_data.csv to Iris_data (1).csv
```

```
iris = pd.read_csv('Iris_data.csv')
iris.shape
col_list = iris.columns
print(type(col_list))
print(col_list[:])
iris['Species'].value_counts()
iris_data = iris.iloc[:,1:5] # select all the rows and col indices 1 to 4
iris_lables = iris.iloc[:,5:] # select all the rows and 5th cloumn
iris_data.shape
iris_data.head(2)
```

```
<class 'pandas.core.indexes.base.Index'>
Index(['row', 'Sepal length', 'Sepal width', 'Petal length', 'Petal width',
       'Species'],
      dtype='object')
```

	Sepal length	Sepal width	Petal length	Petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

```
iris_lables.shape
iris_lables.head(2)
```

	Species
0	I. setosa
1	I. setosa

```
#standardizing using sklearn pre-processing
iris_standard = StandardScaler().fit_transform(iris_data)  # this has transformed dataframe t
#each row in df is a list we will have n inner lists in a outer list,thats why length of iris
#length of each inner list is 4.
print('length of iris_standard is ',len(iris_standard))
print('length of inner list is',len(iris_standard[0]))
print('sample elements are')
print((iris_standard[0:3]))
```

```
length of iris_standard is  150
length of inner list is 4
sample elements are
[[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
 [-1.14301691 -0.13197948 -1.34022653 -1.3154443 ]
 [-1.38535265  0.32841405 -1.39706395 -1.3154443 ]]
```

```
#splitting dataset into train and test

iris_lables_np = iris_lables.values.reshape(1,150)
x_train, x_test, y_train, y_test = train_test_split(iris_standard, iris_lables_np[0], test_si
print(x_test[0:2],y_test[0:2])
print(len(x_test),len(y_test))
print(len(x_train),len(y_train))
```

```
[[ 3.10997534e-01 -5.92373012e-01  5.35408562e-01  8.77547895e-04]
 [-1.73673948e-01  1.70959465e+00 -1.16971425e+00 -1.18381211e+00]] ['I.\xa0versicolor'
50 50
100 100
```

```
#Training using K_NN

neigh = KNeighborsClassifier(n_neighbors=5)
neigh.fit(x_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
#predicting

predict_array = neigh.predict(x_test)
print(metrics.accuracy_score(y_test, predict_array))
#print(y_test[3])
#print(y_test[0])

for i in range(len(predict_array)):
   if (predict_array[i] != y_test[i]):
       print('actual is {} but predicted is {}'.format(y_test[i],predict_array[i]))
```

```
        print('Wrong')
```

```
    0.98
    actual is I. virginica but predicted is I. versicolor
    Wrong
```

```
#prediction on non standardized data
x_train, x_test, y_train, y_test = train_test_split(iris_data, iris_lables_np[0], test_size=0
neigh2 = KNeighborsClassifier(n_neighbors=5)

neigh2.fit(x_train, y_train)
predict_array = neigh2.predict(x_test)
print(metrics.accuracy_score(y_test, predict_array))

print(x_train.shape)
print(y_train.shape)
```

```
    0.98
    (100, 4)
    (100,)
```

```
#cross validation using 10 folds,cv=10
k_list= [1,3,5,7,9]
cv_scores=[]

for i in k_list:
    cross_neigh = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(cross_neigh,x_train, y_train,cv=10)
    cv_scores.append(np.mean(scores))

print(len(cv_scores))
print(cv_scores)

cv_score_zip=zip(k_list,cv_scores)

for i in cv_score_zip:
    print(i)


#plot for K-value and accuracy using 10 fold cv.

plt.figure('Iris_KNN')
plt.xlabel('k-value')
plt.ylabel('cv_score')
plt.grid()
plt.plot(k_list,cv_scores)
plt.show()

# based on above observations we are getting maximum accuracy when k=7,
```

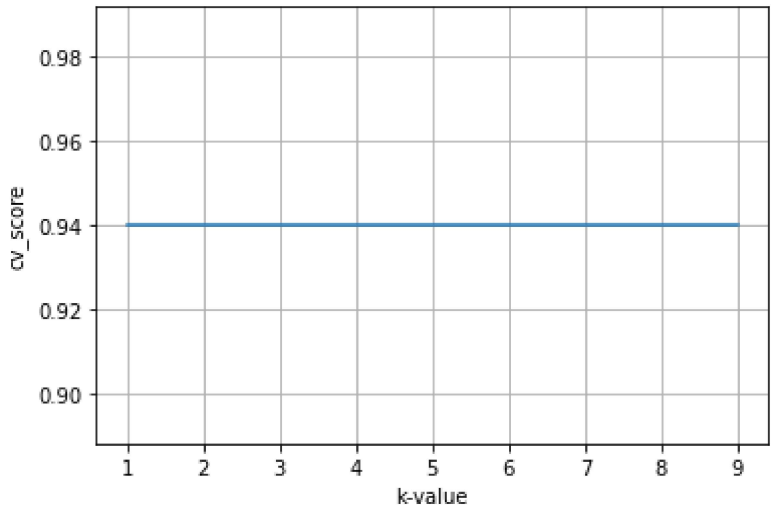
```
#So we will use K-value 7 and predict on test datsset and see accuracy.

neigh_K7 = KNeighborsClassifier(n_neighbors=7)
neigh_K7.fit(x_train, y_train)
predict_array_k7 = neigh_K7.predict(x_test)
print(metrics.accuracy_score(y_test, predict_array_k7))
predict_probability = neigh_K7.predict_proba(x_test)

#zipped_pobability = zip(predict_array_k7,predict_probability)
#for i in zipped_pobability:
#    print(i)

cross_predict = cross_val_predict(cross_neigh,x_test,y_test,cv=10)
print(metrics.accuracy_score(y_test, cross_predict))
```

```
5
[0.9400000000000001, 0.9400000000000001, 0.9400000000000001, 0.9400000000000001, 0.9400(
(1, 0.9400000000000001)
(3, 0.9400000000000001)
(5, 0.9400000000000001)
(7, 0.9400000000000001)
(9, 0.9400000000000001)
```




```
0.98
0.96
```

```
#confusion matrix and classification_report
#precision = TP/TP+FP
#Recall = TP/TP+FN

print(metrics.confusion_matrix(y_test, cross_predict))
print(metrics.classification_report(y_test, cross_predict))
```

```
[[19  0  0]
 [ 0 15  0]
```

```
 [ 0  2 14]]
                precision    recall  f1-score   support

    I. setosa       1.00      1.00      1.00        19
I. versicolor       0.88      1.00      0.94        15
 I. virginica       1.00      0.88      0.93        16

     accuracy                           0.96        50
    macro avg       0.96      0.96      0.96        50
 weighted avg       0.96      0.96      0.96        50
```